

(4)

```
# sip client kludge
implement Command;

Mod : con "sipc";

include "sys.m";
sys: Sys;
stderr : ref Sys->FD;

include "draw.m";

include "daytime.m";

include "csget.m";

daytime: Daytime;

include "sh.m";

Laddr : string;
default_lport : con "5678";
default_rtpport : con "3456";
Calln : int;
active := 0;

init(ctxt : ref Draw->Context, args : list of string)
{
  sys = load Sys Sys->PATH;
  stderr = sys->filides(2);
  daytime= load Daytime Daytime->PATH;
  if(daytime == nil) {
    sys->fprintf(stderr, "sip: load %s: %r\n", Daytime->PATH);
    return;
  }
  Calln = ntime() - int 5e+07;

  cs := load CsGet CsGet->PATH;
  (nil, Laddr, nil) = cs->hostinfo(nil);
  if (Laddr == nil) return;
  sys->print("This address: %s\n", Laddr);

  if (args != nil)
    args = tl args;
    opt : string;
    if (args != nil)
      opt = hd args;
    client : string;
    case opt {
      "?" or "-?" or "help" or "-help" => usage(); return;
      * => {
        if (opt != nil && opt[0] == '$') {
          nc := int opt[1:];
          client = nth(nc, readlist("/services/config/sip_phones"));
          args = tl args;
        }
      }
    }
    if (args != nil && client == nil) {
      client = hd args;
      args = tl args;
    }
    if (client == nil) client = "8090:8090";

    if (args != nil)
      clients = args;
    else readclients();

    ch := chan of int;
    client = thisclient(client);
    sys->print("This client: %s\n", client);
    if (client != nil) {
      (nil, nil, port) := expand(client);
      (ok, conn) := announceudpport(port);
      if (ok < 0) return;
    }
  }
}
```

```

        spawn listen(conn.dfd, ch);
        active = <- ch;
    }
    spawn rcmd(client, ch);
    pid := <- ch;
    #for(l := clients; l != nil; l = tl l)
    #    connect(client, hd l);
}

clients : list of string;

# load the last record of all clients that connected via sip
readclients()
{
    clients = readlist("/services/server/sip_clients");
}

include "kill.m";

cleanup()
{
    if (pid := active) {
        active = 0;
        kp := load Kill Kill->PATH;
        kp->killpid(string pid, array of byte "kill");
    }
}

# /tmp/sipcmd channel to control client from another program
# this does not deal with digit collection yet...

#sipsrv : con "sipcmd";
sipsrv : con "sc";

rcmd(client : string, rch : chan of int)
{
    mp := "/tmp";
    sys->bind("#s", mp, sys->MBEFORE);
    ch := sys->file2chan(mp, sipsrv);
    if (ch == nil) {
        rch <- = 0;
        sys->print(Mod+": file2chan %s/%s %r\n", mp, sipsrv);
        return;
    }
    else rch <- = sys->pctl(0,nil);

    sys->print(Mod+": %s/%s is the command interpreter\n", mp, sipsrv);

    stop := 0;
    while (!stop) {
        alt {
            (o, data, fid, wc) := <- ch.write =>
                if (data != nil && wc != nil) {
                    sys->print(Mod+"> %s\n", string data);
                    stop = sipdo(client, string data);
                    wc <- = (len data, nil);
                }
            (o, n, fid, rc) := <- ch.read =>
                data := array of byte "sip commands - write help to read the
                if (rc != nil && n > 0) {
                    if (n < len data) data = data[0:n];
                    rc <- = (data, "");
                }
                else if (rc != nil) rc <- = (nil, "");
        }
    }
    cleanup();
}

Call : adt
{
    conn : ref Sys->Connection;
    frum : string;
    tu : string;
    callid : string;
}

```

```

    cseq : string;
    state : string;
    session : ref Session;
};

Session : adt
{
    sid    : string;
    data   : string;
    rdata  : list of string;
    audio  : ref Audio;
};

Audio : adt
{
    addr1 : string;
    addr2 : string;
    tipe  : int;
    conn1 : ref Sys->Connection;
    conn2 : ref Sys->Connection;
};

Scall : ref Call;
# only one call for now

sipdo(client, cmd : string) : int
{
    (nil, cl) := sys->tokenize(cmd, " \t\r\n");
    c := Scall;
    if (cl == nil) return 0;
    case hd cl {
        "a" => {
            if (c != nil) {
                if (c.state == "INVITE 180 Ringing") {
                    c.state = "INVITE 200 OK";
                    send(c);
                    return 0;
                } else if (start("INVITE ", c.state)) {
                    nextstate(c);
                    return 0;
                }
            } else {
                sys->print("in call %s\n", c.callid);
                return 0;
            }
        }
        if (tl cl != nil) {
            line := hd tl cl;
            called := findclient(line);
            if (called != nil) Scall = c = connect(client, called);
            else sys->print("client not found at line %s\n", line);
        }
        else sys->print("missing line number\n");
    }
    "z" => {
        if (c == nil) c = Scall = Rcall;
        if (c == nil) sys->print("no current call\n");
        else {
            if (c.state == "INVITE 200 OK") {
                c.state = "ACK";
                send(c);
                return 0;
            }
            else {
                Scall = c = cancel(c);
                Scall = c = nil;
                return 0;
            }
        }
    }
    "q" => return 1;
    * => sys->print("a <number>, z, and q : are supported commands\n");
}
return 0;
}

```

```

findclient(line : string) : string
{
    for(l := clients; l != nil; l = tl l) {
        (num, nil, nil) := expand(hd l);
        if (num == line) return hd l;
    }
    return nil;
}

ntime() : int
{
    return int 1e+09 + daytime->now();
}

rtime() : int
{
    return daytime->now();
}

usage()
{
    sys->print("usage: sip [this_line#:this_port] [remote_line@ripaddr:rport]... [more c
}

thisclient(client : string) : string
{
    if (client == nil)
        for(l := clients; l != nil; l = tl l) {
            (n, la) := sys->tokenize(hd l, "@");
            if (n == 0) {
                client = hd la;
                break;
            }
        }
    (m, lc) := sys->tokenize(client, ":");
    sys->print("client: %s %d\n", client, m);
    if (m > 1)
        return hd lc + "@" + Laddr + ":" + hd tl lc;
    else return client + "@" + Laddr + ":" + default_lport;
}

expand(client : string) : (string, string, string)
{
    (n, la) := sys->tokenize(client, "@:");
    if (n >= 2) {
        line := hd la;
        addr := hd tl la;
        port := default_lport;
        if (n == 3)
            port = hd tl tl la;
        return (line, addr, port);
    }
    return (nil, nil, nil);
}

connect(frum, tu : string) : ref Call
{
    (fline, faddr, fport) := expand(frum);
    (tline, taddr, tport) := expand(tu);
    sys->print("Connect to %s at udp!%s!%s\n", fport, taddr, tport);
    (ok, conn) := dialudpport(taddr, tport, fport);
    if (ok < 0) return nil;
    callid := sid2callid(string ntime());
    return send(ref Call(ref conn, frum, tu, callid, nil, "INVITE", nil));
}

cancel(c : ref Call) : ref Call
{
    c.state = "CANCEL";
    udpaudioend(c.session);
    return send(c);
}

Siptags : list of string;

```

```

siptagp(s : string) : int
{
    if (Siptags == nil) Siptags = "INVITE" :: "ACK" :: "BYE" :: "CANCEL" :: nil;
    for(l := Siptags; l != nil; l = tl l)
        if (start(hd l, s)) return 1;
    return 0;
}

endstatep(s : string) : int
{
    return s == "CANCEL" || s == "BYE" || start("BYE ", s);
}

send(c : ref Call) : ref Call
{
    tag := c.state;
    if (!siptagp(tag)) {
        sys->fprintf(stderr, "Unknown SIP event %s\n", tag);
        return nil;
    }
    phonetags : list of string;
    phonestate : string;
    (nil, ltag) := sys->tokenize(tag, " \t");
    if (ltag != nil && tl ltag != nil) {
        tag = hd ltag;
        phonetags = tl ltag;
        for(l := phonetags; l != nil; l = tl l)
            phonestate += " " + hd l;
    }
    sys->print("current state %s%s\n", tag, phonestate);
    frum := c.frum;
    tu := c.tu;
    callid := c.callid;

    if (c.callid == nil) sys->fprintf(stderr, "Missing callid in call to %s\n", tu);

    (fline, faddr, fport) := expand(frum);
    (tline, taddr, tport) := expand(tu);

    header, data : string;
    if (phonestate == nil) header += tag + " sip:" + tu + ";user=phone ";
    header += "SIP/2.0" + phonestate + "\r\n";
    header += "Via: SIP/2.0/UDP " + faddr + ":" + fport + "\r\n";
    if (phonetags != nil && hd phonetags == "200" && tag == "BYE") {
        header += "From: <sip:" + frum + ">\r\n";
        header += "To: <sip:" + tu + ";user=phone>\r\n";
    }
    else {
        header += "From: " + fline + "_phone<sip:" + frum + ">\r\n";
        header += "To: " + tline + "<sip:" + tu + ";user=phone>\r\n";
    }
    header += "Call-ID: " + callid + "@" + faddr + "\r\n";
    cseq := c.cseq;
    if (cseq == nil || tag == "BYE") {
        seqn := 1;
        if (tag == "BYE") seqn++;
        cseq = string seqn + " " + tag;
        c.cseq = cseq;
    }
    header += "CSeq: " + cseq + "\r\n";

    if (phonetags == nil && tag == "INVITE")
        header += "Subject: Inferno Ephone INVITE\r\nContent-Type: application/sdp\r\n";

    if (phonetags != nil && hd phonetags == "200" && tag == "INVITE")
        header += "Contact: <" + tu + ">\r\nContent-Type: application/sdp\r\n";
    header += "Content-Length: ";

    csp := 0;
    if ((phonetags == nil || hd phonetags == "200") && tag == "INVITE") {
        rtpport := default_rtpport;
        daddr := faddr;
        if (phonetags != nil && hd phonetags == "200") {
            rtpport = string (int rtpport + 10);

```

```

        daddr = taddr;
    }
    sid := callid2sid(callid);
    data += "v=0\r\no=- " + sid + " " + sid + " IN IP4 " + daddr + "\r\n";
    data += "s=Inferno Ephone Session\r\n";
    data += "c=IN IP4 " + faddr + "\r\n" + "t=" + string rtime() + " 0\r\n" + "m=audio " + rtpport + "
    csp = addsession(c, sid, data);
}
msg := header + string len data + "\r\n\r\n" + data;

if (c.conn == nil) {
    sys->print("RE-Connect to %s at udp!%s!%s\n", fport, taddr, tport);
    (ok, conn) := dialudpport(taddr, tport, fport);
    if (ok >= 0) c.conn = ref conn;
}

if (c.conn != nil) {
    fd := c.conn.dfd;
    sys->print("Sending: \r\n%s\r\n", msg);
    sys->fprintf(fd, "%s", msg);
    if (csp) startaudio(c.session, 1);
}
else sys->fprintf(stderr, "Send error: mission connection\n");
return c;
}

addsession(c : ref Call, sid, data : string) : int
{
    if (c.session == nil)
        c.session = ref Session(sid, data, nil, nil);
    else {
        s := c.session;
        if (s.sid != nil && sid != nil && s.sid != sid) {
            sys->fprintf(stderr, "changing session id %s->%s\n", s.sid, sid);
            s.sid = sid;
        }
        if (s.data == nil) s.data = data;
        else s.rdata = data :: s.rdata;
        return 1;
    }
    return 0;
}

startaudio(s : ref Session, tipe : int)
{
    m1 := retrieve("m=", s.data);
    c1 := retrieve("c=", s.data);
    m2, c2 : string;
    sys->print("session %s data audio:\n\t%s\n", s.sid, m1);
    if (s.rdata != nil) {
        sys->print("\trdata audio:\n");
        for (l := s.rdata; l != nil; l = tl l) {
            m2 = retrieve("m=", hd l);
            c2 = retrieve("c=", hd l);
            sys->print("\t:: %s\n", m2);
        }
        if (m2 != nil)
            udpaudiocall(s, tipe, snth(2, c1), snth(1, m1), snth(2, c2), snth(1,
    }
}

udpaudiocall(s : ref Session, tipe : int, faddr, fport, taddr, tport : string)
{
    sys->print("\n----> Start UDP audio: %d %s:%s %s:%s\n\n", tipe, faddr, fport, taddr,
    s.audio = ref Audio(faddr + ":" + fport, taddr + ":" + tport, tipe, nil, nil);
}

udpaudioend(s : ref Session)
{
    if (s == nil) return;
    a := s.audio;
    if (a != nil)
        sys->print("\n----> Stop UDP audio: %d %s %s\n\n", a.tipe, a.addr1, a.addr2);
    s.audio = nil;
}

```

```

Idkey : con 22e+07;
sid2callid(sid : string) : string
{
    return string (int sid - int Idkey);
}

callid2sid(cid : string) : string
{
    return string (int cid | int Idkey);
}

Invite0 : con "INVITE sip:8089@135.2.180.21:8089:5060;user=phone SIP/2.0\r\nVia: SIP/2.0/UDP
InvData0 : con "v=0\r\no=- 1907994094 1907994094 IN IP4 135.2.180.20\r\ns=VOVIDA Session\r\nr
invite0_test(fd : ref Sys->FD)
{
    n := len InvData0;
    s := Invite0+string n+"\r\n\r\n"+InvData0;
    sys->print("Sending: %s\r\n", s);
    sys->fprintf(fd, "%s", s);
}

announceudpport(port : string) : (int, Sys->Connection)
{
    addr := "udp!*!" + port;
    (ok, conn) := sys->announce (addr);

    if (ok < 0) {
        sys->fprintf(stderr, "Cannot announce at port %s \n", addr );
        return (ok, conn);
    }

    # open the data file for the connection
    conn.dfd = sys->open (conn.dir+"/data", sys->ORDWR);

    if (conn.dfd == nil){
        sys->fprintf(stderr, "Cannot open file %s/data\n", conn.dir);
        return (-1, conn);
    }

    sys->print("Announced port %s\n", port);
    return (ok, conn);
}

Rcall : ref Call;
# only one received call for now

listen(fd : ref Sys->FD, ch : chan of int)
{
    ch <- = sys->pctl(0, nil);
    buf := array[1024] of byte;
    while(active) {
        n := sys->read(fd, buf, len buf);
        if (n < 0) return;
        if (n > 0) {
            csp := 0;
            sys->print("Receiving:\n");
            (hl, data) := decode(string buf[0:len buf]);
            c := mkcall(hl, data);
            if (Scall != nil)
                if (c.callid == Scall.callid) {
                    Scall.state = c.state;
                    if (c.session != nil)
                        csp = addsession(Scall, c.session.sid, c.session.si
                    c = Scall;
                }
            else if (Rcall != nil)
                if (c.callid == Rcall.callid) {
                    Rcall.state = c.state;
                    if (c.session != nil)
                        csp = addsession(Rcall, c.session.si
                    c = Rcall;
                }
        }
    }
}

```

```

        else {
            sys->print("Switching received calls %s->%s\
c.conn = Rcall.conn;
Rcall = c;
        }
        else Rcall = c;
    else
        Scall = c;

    if (Scall != nil && endstatep(Scall.state)) {
        c = Scall;
        nextstate(c);
        udpaudioend(c.session);
        Scall = c = nil;
    }
    if (Rcall != nil && endstatep(Rcall.state)) {
        c = Rcall;
        nextstate(c);
        udpaudioend(c.session);
        Rcall = c = nil;
    }
    if (c != nil && c.conn == nil) {
        (fline, faddr, fport) := expand(c.frum);
        (tline, taddr, tport) := expand(c.tu);
        tport = default_lport; #override
        sys->print("Connect BACK to %s at udp!%s!%s\n", tport, faddr
(ok, conn) := dialudpport(faddr, fport, tport);
        if (ok >= 0) c.conn = ref conn;
        else sys->print("Connect failed\n");
    }
    if (c != nil) {
        Scall = Rcall = c;
        if (csp) startaudio(c.session, 0);
        nextstate(c);
    }
}

}

nextstate(c : ref Call)
{
    case c.state {
        "INVITE" => {
            c.state += " 180 Ringing";
            send(c);
        }
        "INVITE 180 Ringing" => {
            c.state = "INVITE 200 OK";
            send(c);
        }
        "INVITE 200 OK" => {
            c.state = "ACK";
            send(c);
        }
        "ACK" => {
            c.state = "BYE";
            send(c);
        }
        "BYE" => {
            c.state += " 200 OK";
            send(c);
        }
        * => sys->fprintf(stderr, "waiting state %s\n", c.state);
    }
}

mkcall(l : list of string, data : string) : ref Call
{
    (nil, ll) := sys->tokenize(hd l, " \t");
    state, substate : string;
    if (ll != nil) {
        state = hd ll;
        for(ll = tl ll; ll != nil; ll = tl ll)
            substate += " "+ hd ll;
    }
}

```



```

cseq := findval("CSeq:", l);
(nil, ll) = sys->tokenize(cseq, " \t");
if (ll != nil) {
    if (start("SIP/", state)) {
        if (tl ll != nil)
            state = hd tl ll;
    }
    cseq = l2string(ll);
}
if (!start(" sip:", substate))
    state += substate;

frum := sipurlval(findval("From:", l));
tu := sipurlval(findval("To:", l));
callid := findval("Call-ID:", l);
if (callid != nil) {
    (nil, ll) = sys->tokenize(callid, " \t@");
    if (ll != nil) callid = hd ll;
}
sid : string;
if (data != nil) {
    p1 := find("o=-", data); if (p1 < 0) p1 = 0; else p1 += len "o=-";
    p2 := poso('\n', data, p1); if (p2 < 0) p2 = 0;
    (nil, ll) = sys->tokenize(data[p1:p2], " \t\r\n");
    if (ll != nil) sid = hd ll;
}
s : ref Session;
if (sid != nil)
    s = ref Session(sid, data, nil, nil);
return ref Call(nil, frum, tu, callid, cseq, state, s);
}

sipurlval(s : string) : string
{
    su := "<sip:";
    p1 := find(su, s);
    if (p1 < 0) return nil;
    else p1 += len su;
    p2 := poso('>', s, p1); if (p2 < 0) return nil;
    rs := s[p1:p2];
    if (rs != nil) {
        (nil, l) := sys->tokenize(rs, ";");
        sys->print("sipurl: %s\n", hd l);
        return hd l;
    }
    return rs;
}

decode(s : string) : (list of string, string)
{
    r : list of string;
    data : string;
    p, pn, n : int = 0;
    while ((p = poso('\r', s, n)) >= 0 || (pn = poso('\n', s, n)) >= 0) {
        if (pn) p = pn;
        if (p > n) r = s[n:p] :: r;
        sl := getval("Content-Length:", s[n:p]);
        nc := '\n';
        if (pn) {
            nc = '\r';
            pn = 0;
        }
        if (len s > p+1 && s[p+1] == nc) p++;
        sys->print("%s", s[n:p+1]);
        n = p = p + 1;
        if (sl != nil) {
            l := int sl;
            data = s[n:n+l];
            sys->print("%s\r\n\r\n", data);
            break;
        }
    }
    return (reverse(r), data);
}

```

```

dialudpport(addr, rport, port : string) : (int, Sys->Connection)
{
  (ok, conn) := sys->dial("udp!" + addr + "!" + rport, port);
  if (ok < 0)
  {
    sys->fprintf (stderr, "Cannot connect to udp!%s!%s local %s \n", addr, rport, port);
    return (ok, conn);
  }
  sys->print("New connection to udp!%s!%s local %s\n", addr, rport, port);
  return(ok, conn);
}

# string and list utils

l2string(l1 : list of string) : string
{
  r : string;
  for(; l1 != nil; l1 = tl l1) {
    r += hd l1; if (tl l1 != nil) r += " ";
  }
  return r;
}

snth(n: int, s : string) : string
{
  (nil, l) := sys->tokenize(s, " \t\r\n");
  return nth(n, l);
}

nth(n: int, l : list of string) : string
{
  for(i := 0; l != nil; l = tl l) {
    if (i == n) return hd l;
    i++;
  }
  return nil;
}

retrieve(k, s : string) : string
{
  p := find(k, s);
  if (p >= 0) {
    z := poso('\r', s, p);
    if (z < p) z = poso('\n', s, p);
    if (z < p) z = len s;
    return s[p:z];
  }
  return nil;
}

find(e, s : string) : int
{
  for(i := 0; i < len s - len e; i++) {
    ok := 1;
    for (j := 0; j < len e; j++)
      if (e[j] != s[i+j]) {ok = 0; break;}
    if (ok) return i;
  }
  return -1;
}

findval(k : string, l : list of string) : string
{
  r : string;
  for(; l != nil; l = tl l)
    if ((r = getval(k, hd l)) != nil) break;
  return r;
}

reverse(l : list of string) : list of string
{
  r : list of string;
  for(; l != nil; l = tl l) r = hd l :: r;
  return r;
}

```

```

poso(c : int, s : string, o : int) : int
{
    for(i := 0; i < len s; i++)
        if (s[i] == c) return i;
    return -1;
}

start(k, s : string) : int
{
    if (len s >= len k && k == s[0:len k])
        return 1;
    return 0;
}

getval(k, s : string) : string
{
    if (len s >= len k && k == s[0:len k])
        return s[len k:];
    return nil;
}

# Read list from file

readlist(path : string) : list of string
{
    (ok, dir) := sys->stat(path);
    if (ok < 0) {
        sys->fprintf(stderr, "stat: %s: %r\n", path);
        return nil;
    }
    shfd := sys->open(path, sys->OREAD);
    if (shfd == nil) {
        sys->fprintf(stderr, "open: %s: %r\n", path);
        return nil;
    }
    lc := dir.length;
    if (lc == 0) return nil;

    buf := array[lc] of byte;
    m := 0; n := lc;
    while ((n = sys->read(shfd, buf[m:], lc - m)) > 0)
        m += n;
    if (n < 0) {
        sys->fprintf(stderr, "read: %s: %r\n", path);
        if (!m) return nil;
    }
    # sys->print("buf[%d]=%s\n", m, string buf);
    (nil, r) := sys->tokenize(string buf[0:m], " \t\r\n");
    return r;
}

```